D&D No . 08223/100S137-US3

METHOD AND SYSTEM FOR REAL-TIME TAMPER EVIDENCE GATHERING FOR SOFTWARE

Related Applications

This application claims the benefit of U.S. Provisional Application No. 60/412,265 filed on September 20, 2002, which is hereby claimed under 35 U.S.C. §119(e).

Field of the Invention

The present invention relates generally to verifying software functional or procedural integrity, and determining whether a software process is not functioning normally.

Background of the Invention

The software industry relies on virtually billions of lines of software to work as expected. Companies write code, sell code, and rely on processes embodied in the code that is distributed. Unfortunately, once the code leaves the control of the developing company, the code can often be changed or used outside of the context that it was intended.

For example, some software vendors distribute products for playing video or audio files on a user's computer. Some of these packages have internal protection mechanisms that allow video or audio to be protected when used in conjunction with this product. Unfortunately, if a user inserts their own code into the product, or runs another piece of software that captures the computer screen as a movie, they can obtain a copy of the content being played. This is not something that most vendors would like to see occur. Unfortunately, this action currently is outside the context of vendor's product and cannot be easily detected using available technology.

Thus, it is with respect to these considerations and others that the present invention has been made.

Brief Description of the Drawings

Non-limiting and non-exhaustive embodiments of the present invention are described with reference to the following drawings. In the drawings, like reference numerals refer to like parts throughout the various figures unless otherwise specified.

For a better understanding of the present invention, reference will be made to the following Detailed Description of the Preferred Embodiment, which is to be read in association with the accompanying drawings, wherein:

FIGURE 1 illustrates an exemplary environment in which the present invention may be practiced;

FIGURE 2 illustrates a flow diagram showing one embodiment for a process of determining real-time tamper evidence of a software process;

FIGURE 3 illustrates an example of a general Chaos Game Representation (**CGR**) plot with a possible pattern for a consequence vector, **Ap**;

FIGURE 4 illustrates an example of a possible pattern vector, **Pp**, where system calls frequencies are substantially equal; and

FIGURE 5 illustrates one embodiment of a computer in which a tamper detector operates, in accordance with the present invention.

Detailed Description of the Preferred Embodiment

In the following detailed description of exemplary embodiments of the invention, reference is made to the accompanied drawings, which form a part hereof, and which is shown by way of illustration, specific exemplary embodiments of which the invention may be practiced. Each embodiment is described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized, and other changes may be made, without departing from the spirit or scope of the invention. The following detailed description is, therefore, not to be taken in a limiting sense.

Throughout the specification, the term "connected" means a direct connection between the things that are connected, without any intermediary devices or components. The term "coupled" means a direct connection either between the things

that are connected, or an indirect connection through one or more passive or active intermediary devices or components. The meaning of "a," "an," and "the" include plural references. The meaning of "in" includes "in" and "on."

The present invention is directed at providing a method and system for differentiating between normal operational characteristics and abnormal (also called non-normal) operational characteristics such that software product tampering may be identified programmatically. Additionally, the invention provides for a variance for identification of behavior that is defined to be within the realm of normal user behavior.

It has been determined that abnormal behavior may be considered that behavior that is not normal behavior. Detection of normal behavior provides for the ability to detect abnormal behavior. Data is gathered and when a sufficient amount of abnormal behavior has been detected, a signal may be provided such that any of a variety of actions may be performed. Such actions may include, but is not limited to, providing an alert message to a software provider, providing a warning message, shutting down a computing device, software process, and the like.

The invention obtains samples of predetermined traits needed to monitor the software for evidence of tampering. In most cases, this equates to a select number of system level calls that access resources that may be considered important, such as reading and writing to hard drives, memory, network resources, and the like.

Each predetermined trait is assigned a unique number. When a piece of software is running, it produces a stream of data identifying when predetermined traits that need to be monitored are utilized. Each predetermined trait is summarized from the data and statistical information about a trend associated with each trait may be produced.

The trends of the predetermined traits are compared to identified good trends to determine if they are normal. If there is not enough data to determine the trend of the traits exhibited, the result will be that the behavior is unknown. When there is enough data to make a determination, then the result may be normal or abnormal.

Operating Environment

Figure 1 shows a schematic block diagram of an exemplary system-operating environment that benefits from the teachings of the present invention. The system includes a server 101, a client 103, and a network 102. The client 103 is coupled to the network, and the server is coupled to the network. The client 103 includes an example application 104 that the integrity is to be determined.

The client 103 may be a computing device, such as portable computer, desktop computer, personal digital assistant (PDAs), a media player, or other similar device that a software application may be exposed to tampering. One embodiment of client 103 is described in more detail below, in conjunction with FIGURE 5. Similarly, the server 101 may be any of a variety of similar devices that is arranged to communicate through the network 102 to the client 103.

The network 102 can employ any form of computer readable media for communicating information from one electronic device to another. Also, network 102 can include the Internet in addition to local area networks (LANs), wide area networks (WANs), direct connections, such as through a universal serial bus (USB) port, other forms of computer-readable media, or any combination thereof. On an interconnected set of LANs, including those based on differing architectures and protocols, a router acts as a link between LANs, enabling messages to be sent from one to another. Also, communication links within LANs typically include twisted wire pair or coaxial cable, while communication links between networks may utilize analog telephone lines, full or fractional dedicated digital lines including T1, T2, T3, and T4, Integrated Services Digital Networks (ISDNs), Digital Subscriber Lines (DSLs), wireless links including satellite links, or other communications links known to those skilled in the art. Furthermore, remote computers and other related electronic devices can be remotely connected to either LANs or WANs via a modem and temporary telephone link. A remote computer may act in a number of ways, including as a WWW (content) server or a client with a browser application program.

In Figure 1, the client 103 is in communication with the network 102 and provides for transmitting application 104 user profiles, software process behavioral information, trait data, and the like, to the server 101. However, the present invention is

not limited to network connectivity. For example, the verification of the integrity of application 104 may be performed by components on client 103 without requiring network connectivity.

Therefore, the operating environment shown in FIGURE 1 is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use, or functionality of the invention. Other well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

FIGURE 5 illustrates one embodiment of a computer in which a tamper detector operates according to one embodiment of the invention. Computer 500 illustrates one embodiment of client 103 shown in FIGURE 1. Computer 500 may also be employed to illustrate one embodiment of server 101 shown in FIGURE 1. Computer 500 may include many more components than those shown. The components shown, however, are sufficient to disclose an illustrative embodiment for practicing the invention.

Computer 500 includes processing unit 512, video display adapter 514, and a mass memory, all in communication with each other via bus 522. The mass memory generally includes RAM 516, ROM 532, and one or more permanent mass storage devices, such as hard disk drive 528, tape drive, optical drive, and/or floppy disk drive. The mass memory stores operating system 520 for controlling the operation of computer 500. Any general-purpose operating system may be employed. Basic input/output system ("BIOS") 518 is also provided for controlling the low-level operation of computer 500.

As illustrated in FIGURE 5, computer 500 also can communicate with the Internet, or some other communications network, such as network 102 in FIGURE 1, via network interface unit 510, which is constructed for use with various

communication protocols including the TCP/IP protocol. Network interface unit 510 is sometimes known as a transceiver or transceiving device.

The mass memory as described above illustrates another type of computer-readable media, namely computer storage media. Computer storage media may include volatile, nonvolatile, removable, and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. Examples of computer storage media include RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by a computing device.

In one embodiment, the mass memory stores program code and data for implementing operating system 520. The mass memory may also store additional program code and data for performing the functions of computer 500, and tamper detector 502. One or more applications, such as application 104 when computer 500 is employed to illustrate client 103 of FIGURE 1, tamper detector 502, and the like, may be loaded into mass memory and run on operating system 520.

While computer 500 illustrates tamper detector 502 as a component, the present invention is not so limited. For example, tamper detector 502 may operate within a server, such as server 101 in FIGURE 1. Tamper detector 502 may also be decomposed into several components, some of which operate within a server, and other components may operate within a client, without departing from the scope of the present invention.

Computer 500 may also include an SMTP handler application for transmitting and receiving e-mail, an HTTP handler application for receiving and handing HTTP requests, and an HTTPS handler application for handling secure connections. The HTTPS handler application may initiate communication with an external application in a secure fashion.

Computer 500 also includes input/output interface 524 for communicating with external devices, such as a mouse, keyboard, scanner, or other input devices not shown in FIGURE 5. Likewise, computer 500 may further include additional mass storage facilities such as CD-ROM/DVD-ROM drive 526 and hard disk drive 528. Hard disk drive 528 is utilized by computer 500 to store, among other things, application programs, databases, and the like.

Generalized Operation

Described below is one embodiment of a process flow for employing graphical representations, such as chaos game representations, and the like, for evaluating evidence of tampering of software.

The process begins by determining the main components related to the application processes of interest.

Virtually every application process may be considered as a set of system calls, which include organized system sequences, and which may be ranked by a limited length. When comparing two different data sets, where one represents the pattern and the other data set represents a prototype, the process employs two major components of that data, Consequence and Frequency. Comparing the two different data sets includes a comparison of consequences and frequencies of certain system calls. However, the present invention is not limited to consequence and frequency, and other components of the data may be employed without departing from the scope of the invention.

The terms "pattern," "pattern data," and the like, refer to a set of data that represents a behavior that is determined to be normal for a given software process, application, and the like. The terms, "prototype," "prototype data," and the like refer to another set of data that represents a behavior of the given software process, application, and the like. Prototype data may be compared to the pattern data to determine whether the software process is considered to have been tampered with.

Creating a fingerprint as a consequence pattern

The process begins by calculating consecutive data for a sequence of file system calls for what is considered to be an unhacked (normal) application process. Various approaches related to graphing representations, and the like, may be employed for analyzing the data. In one embodiment Chaos Game Representation (CGR) is employed to convert the calculated data to CGR dot-data. The obtained CGR dot-data is then converted to the radius-vector data as a fingerprint.

Vector analysis rules are employed for processing the data. Newly created radius vectors are substituted with just one as a pattern from the fingerprint. The pattern is then trained for better matching the real situation.

When the created vector becomes saturated or matured, the **X** and **Y** coordinates of the ending point of the created vector (the origin point of the created vector already has (0, 0) coordinates assigned by default) are retained. A general condition that may indicate that the vector is saturated and mature includes the event when a difference between two measurements of sequence errors, ER1 and ER2, for substantially the same pattern or prototype is equal to or less then a preliminary assigned value Val:

$$| ER1 - ER2 | <= Val$$

Next, the process verifies the vector stability through the next 50 – 100 new system calls. When the vector stability is verified, the number of system calls is assigned to one vector, and the process starts again at creating new dot-data until the new vector is generated. Once the stable vector has been processed, the statistics are reinitialized in preparation for the next vector. By continuing the process, a sequence of vectors is created, which represent all changes in the sequence of the data that have occurred.

Creating a fingerprint as a frequency pattern

A fingerprint for the frequency pattern data may be obtained by calculating a sum of substantially every type of system call that appears in the data processing as well as the total number of system calls that are made.

A vector is created by adding up all of the average directional vectors. FIGURE 3 illustrates an example of a general Chaos Game Representation (**CGR**) plot with a possible pattern for a consequence vector.

The maximal range of such determined values of system calls as Write, Read, Seek, or the like is obtained. These are retained as pattern values.

The process continues to create the new data until the next saturated and mature vector representing the consequence pattern appears. Continuing the process, the sequence of patterns is created, which represent substantially all changes of frequency in the data that happened virtually in real time to each kind of the system calls.

Creating fingerprints in real time as the prototypes to consequence and frequency patterns

The process continues by calculating the consequence data of the file system calls and frequency numbers for the tested application virtually in real time. The process steps described above for the creating of consequence and frequency patterns are continued.

Decision-making

The process next creates a way of analyzing and comparing the real time data to the pattern data. The process runs the decision engine to obtain the results. Each pattern, created as described above, is created separately for the applications of any known kinds and any known computer platforms the user might use and for which the integrity is sought.

Figure 2 illustrates the stages for the four independent processes that are employed for obtaining the results.

Consequence pattern and prototype creation

Initially, the Chaos Game Representation (**CGR**) scatter plot is determined. Next, the process creates the **X** and **Y** coordinates axis through the center of the circle. An assumption may be made that virtually every point is considered as a radius vector representative with the origin (0, 0) point.

The set of the created vectors is considered as a fingerprinting field for the patterns. To create a pattern-vector **Ap** and keep it in the unit circle range the process determines the average sum of all vectors.

Aj (j = 1,N) created on the **CGR**:

$$\mathbf{A_p = A_1 + A_2 + A_3 + \ldots + A_j + \ldots + A_N \equiv 1/N} \sum_{j=1}^{N} Aj$$

where **N** is a number of all points in the **CGR**.

In coordinates form, the **Ap** vector is:

$$\mathbf{A_p\{X_p, Y_p\}}$$

where $\mathbf{X_p = 1/N} * \sum_{j=1}^{N} Xj$ **and** $\mathbf{Y_p = 1/N} * \sum_{j=1}^{N} Yj$.

The new obtained vector $\mathbf{A_p}$ inherits the information from each of the **N** vectors. The previously created vectors (or points) are substituted with one newly created vector $\mathbf{A_j}$ with $\mathbf{X_p}$ and $\mathbf{Y_p}$ coordinates. The new vector, $\mathbf{A_p}$, accumulates substantially all sets of behaviors represented by the **CGR**. Thus, only one radius vector coordinates are retained for each created pattern rather than thousand of points. This improves the efficiency of memory usage and software performance. The process of obtaining the pattern-vector requires fewer computer operations such as additions.

Frequency pattern and prototype creation

The process for creating frequency patterns or prototypes is described above. The process calculates and cumulates the sum for virtually each type of happened system calls for each sample of given length.

A **CGR** is created employing the following process. Typically, the number of points **K** that are equal to or less than the number of possible or chosen system calls is employed. Virtually every new calculated point for virtually each kind of system call may be located on the line between center and a point on the circle surface that represent this kind of system call. Virtually every point represents the vector vertices at the given direction (same as an angle).

The length $\mathbf{L_1}$ of the newly created vector represents the frequency of its appearance and can be determined from the formula:

$$\mathbf{L_1 = 1/TNC} * \sum_{i=1}^{M} Pi$$

where **TNC** is a total number of system calls, $\mathbf{P_i}$ is a single point, and **M** is a total number of calculated points for this type of system call.

Furthermore,

$$\sum_{i=1}^{K} Li <= \mathbf{1}$$

In creating a pattern vector **Pp** (employing substantially the same approach for the prototype vector) an average sum of all vectors **Pj (j = 1, K)** created on the **CGR** is determined:

$$\mathbf{P_p = P_1 + P_2 + P_3 + \ldots + P_j + \ldots + P_K} \equiv \mathbf{1/K} \sum_{j=1}^{K} Pj$$

where **K** is a number of all system calls (vectors) in the **CGR**. In the coordinate form, the **Pp** vector substantially is:

$$\mathbf{P_p\{X_p, Y_p\}}$$

where $\mathbf{X_p} = 1/K^* \sum_{j=1}^{K} Xj$ and $\mathbf{Y_p} = 1/K^* \sum_{j=1}^{K} Yj$

The newly obtained vector $\mathbf{P_p}$ inherits the information from the **K** vectors. FIGURE 4 illustrates an example of a possible pattern vector, **Pp**, where system calls frequencies are substantially equal. In the example shown in FIGURE 4, **K**=5.

Determining Normal from Non-normal Behavior

When incoming data with real time points needs to be processed, the fingerprinting fields of the consequence vector-prototype and the frequency vector-prototype are produced. These are compared to the corresponding vectors-pattern. Analyzing the results, a final decision may be determined.

The process, which is virtually the same for all comparisons, is given below.

Two different vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ are compared by their norms and angle between the vectors. The formula for determining the angle between the two given vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ can be represented as the following:

$$\cos(Ap, Bpr) = (X_1 * X_2 + Y_1 * Y_2) / (|A_p| * |B_{pr}|) =$$

$$= (X_1 * X_2 + Y_1 * Y_2) / (\sqrt{X_1^2 + Y_1^2} * \sqrt{X_2^2 + Y_2^2})$$

Consideration is typically given to the upper (nominator) part of the given fraction because it has practical influence on the preliminary results of analysis and more weight for the further decisions. For example, the following three situations may arise, where the nominators could be as the following:

$X_1 * X_2 + Y_1 * Y_2 = 0$ 1.

$X_1 * X_2 + Y_1 * Y_2 < 0$ 2.

$$X_1 * X_2 + Y_1 * Y_2 > 0 \qquad 3.$$

Equation 1 above illustrates that an angle between vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ is **90** degrees and vectors are perpendicular.

Equation 2 above illustrates that vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ have an opposite direction.

Equations 1 and 2 thus illustrate that substantial abnormality, and as such are ready for a decision to be determined. Equation 3 above illustrates a final decision, and is explained below.

One approach for determining normal from non-normal behavior with respect to equation 3 above that enables reasonable and confident results is described next.

The approach begins by determining a norm $\mathbf{N_1}$ of vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ sum:

$$\mathbf{N_1} = |\mathbf{A_p} + \mathbf{B_{pr}}| = \sqrt{(X_1 + X_2)^2 + (Y_1 + Y_2)^2}\,.$$

Next, a norm $\mathbf{N_2}$ of vectors $\mathbf{A_p}$ and $\mathbf{B_{pr}}$ difference is determined as:

$$\mathbf{N_2} = |\mathbf{A_p} - \mathbf{B_{pr}}| = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}\,.$$

It is then determined how vectors **Ap** and **Bpr** are similar by using their directions:

$$\mathbf{E_1} = \mathbf{1} - (\mathbf{N_1} - \mathbf{N_2}) / \mathbf{N_1} = \mathbf{N_2} / \mathbf{N_1}.$$

The norm **Np** and **Npr** of vectors **Ap** and **Bpr** are determined by:

$$\mathbf{N_p} = \sqrt{X_1^2 + Y_1^2}$$

$$N_{pr} = \sqrt{X_2^2 + Y_2^2}.$$

Similarity of vectors **Ap** and **Bpr** may be determined by employing their lengths, using the following equation:

$$E_2 = | N_p - N_{pr} | / N_p.$$

One way to determine a total difference, **ER**, between vectors **Ap** and **Bpr** may also be represented as:

$$ER = \max (E_1, E_2).$$

The total difference may also be represented by a percentage measurement, as:

$$ER\% = \max (E_1, E_2)*100\%.$$

Next, a Confidence level, **CL**, or fitting Probability, **FP**, may be determined by:

$$CL = FP = 1 - ER.$$

Similarly, a percentage of the confidence level may be determined as:

$$CL\% = FP\% = 100\% - ER\%.$$

Now, if the value for the confidence level shows that the maximum difference between vectors **Ap** and **Bpr**, and their direction, can be trusted, then a determination may be made as whether the process's behavior is normal or non-normal.

For determining a final result, a comparison between one of a *priori* set values, mentioned above, and one of obtained set values during the newest calculation such as **ER, or ER% or CL, or FP, or CL%, or FP%**, may be performed.

A degree of trust may be obtained for the result based in part on the maximal error. Thus, for example, if the maximal error calculated is inside or on the border of the preliminary assigned error interval, the result may be determined to indicate normal behavior of the software process.

When the results of all comparisons (say 2 x 3 = 6) are obtained, then the process creates the procedure for the final decision.

While the above disclosure employed Chaos Game Representations, those skilled in the art will recognize that the invention is not limited to such implementation and other graphical representation schemes may be employed without departing from the scope or spirit of the invention.

The above specification, examples, and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.